KLAST plugin
for CLC Bio softwares

User manual

© 2013 – Korilog SARL

Table of Content

1.Introduction..... 3
2.Algorithm..... 3
3.Setting up a KLAST job..... 5
4.Analysing results..... 8
5.Exporting results..... 9
6.KLAST parameters..... 10
Basic search parameters..... 10
Optimizing search job using KLAST specific arguments..... 11
7.Filtering KLAST results..... 13
Introduction 13
Creating a data filter..... 14
Managing filters..... 15
Using regular expressions..... 16
8.Citing KLAST..... 18
9.References..... 18
Copyright..... 18

1. Introduction

KLAST is a fast, accurate and NGS scalable bank-to-bank sequence similarity search tool providing significant accelerations of seeds-based heuristic comparison methods, such as the Blast suite of algorithms.

Relying on unique software architecture, KLAST takes full advantage of recent multi-core personal computers without requiring any additional hardware devices.

KLAST being designed for bank-to-bank sequence comparisons, it is not appropriate to use on a « per-sequence » basis. In other words, compare your query sequences in batch mode using set of sequences instead of running several KLAST jobs, each of them using a single query sequence.

2. Algorithm

KLAST high-performance sequence similarity search tool relies on a new optimized and combined implementation of the Parallel Local Sequence Alignment Search Tool (PLAST) [1] and the Ordered Index Seed (ORIS) [2] algorithms developed by Inria's Genscale research team [3].

KLAST aims at comparing two large sets of sequences. Query and subject can be both proteins or nucleotides for which you'll use KLASTp or KLASTn to compare them, respectively. Considering proteins versus translated nucleotides, you'll use one of KLASTx, tKLASTx or tKLASTn. KLASTp, KLASTx, tKLASTn and tKLASTx algorithms are actually using the same algorithm, relying on KLASTp principles. KLASTn uses a slightly different approach. The following sections describes KLASTp and KLASTn.

KLASTp aims to compare two large sets of sequences. It implements a three-step seed-based algorithm: (1) indexing, (2) ungap extension and (3) gap extension. Compared to BLASTp algorithm, KLASTp differs essentially during step 1. Steps 2 and 3 are similar between the two algorithms.

1. Both sets are first indexed with subset-seeds of W characters [4]. The index is a table of K entries (K is the number of different seeds). Each entry is associated with a list L of P positions corresponding to all occurrences of the seeds in the sequence set.

2. For each seed, P1 x P2 ungap extensions are performed. P1 (resp. P2) represents the number of elements associated to the list L1 (resp. L2) for the selected seed. Ungap extensions are computed on subsequences made of 2L+W characters (size of the seed + L neighboring residues before and after the seed). Ungap alignments with a score greater than a predefined threshold value are selected for the 3rd step.

3. As the previous step can generate many ungap alignments belonging to the same gap alignment, a checking on the final set of alignments is done before launching the gap process. Dynamic programming algorithm is used to compute final gapped sequence alignments.

Moreover, KLASTp is optimized for multicore processor architectures. The algorithm is natively parallelized following a coarse-grained approach scaling well with the number of cores. Furthermore, gap and ungap extension computation are vectorized using the SSE instruction set of processors.

KLASTn compares two large sets of DNA sequences. Similarly to KLASTp, two indexes including neighboring information are built, allowing the first steps of the search to be fastened. Furthermore, KLASTn includes a new search engines based on the concept of Ordered Seed Indexing [2]. This algorithm suppresses numerous redundant computations during the extension step of alignment construction.

As KLASTp, KLASTn is optimized for today multicore microprocessor architectures, and scale well with the number of cores.

It is worth noting that KLAST uses a Karlin-Altschul statistical model, as BLAST does. As a consequence, every score, e-value, etc. computed by KLAST has the same meaning as BLAST-computed values.

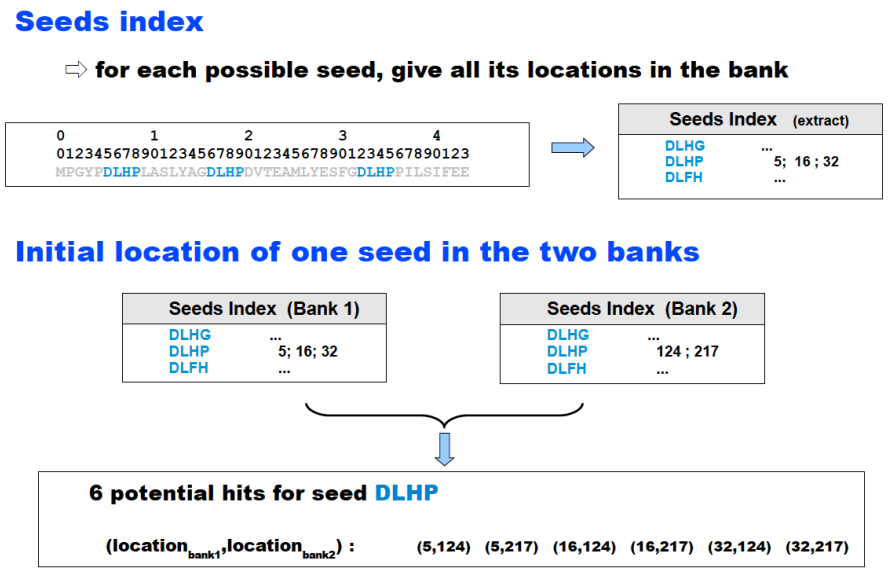

Seeds index
for each possible seed, give all its locations in the bank
0 1 2 3 4
01234567890123456789012345678901234567890123
MPGYPDLHPLASLYAGDLHPDVTEAMLYESFGDLHPPILSIFEE
Seeds Index (extract)
DLHG ...
DLHP 5; 16 ; 32
DLFH ...
Initial location of one seed in the two banks
Seeds Index (Bank 1)
DLHG ...
DLHP 5; 16; 32
DLFH ...
Seeds Index (Bank 2)
DLHG ...
DLHP 124 ; 217
DLFH ...
6 potential hits for seed DLHP
(location bank1 ,location bank2) : (5,124) (5,217) (16,124) (16,217) (32,124) (32,217)

Figure 1 – KLAST algorithm: use of the seeds index to match query and target sequences (bank 1 and bank 2).

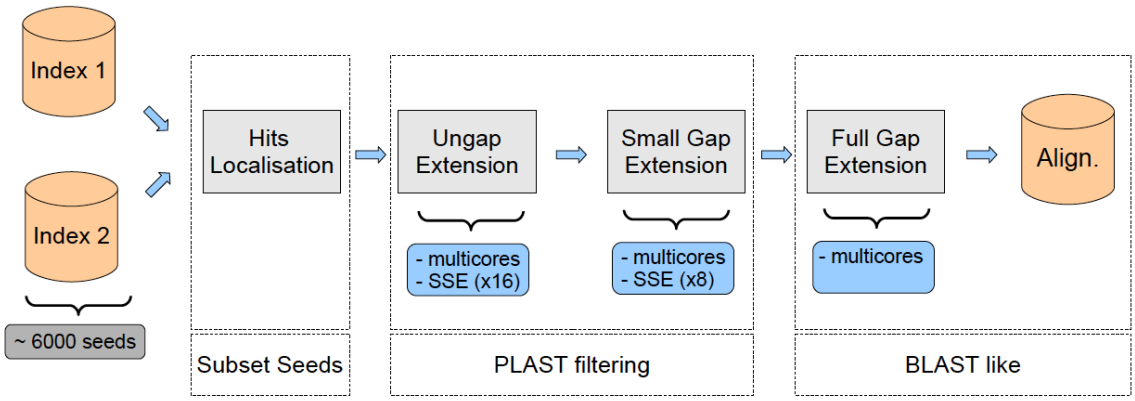
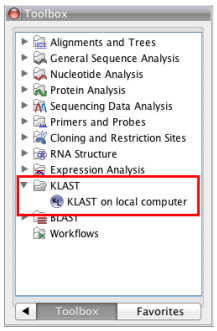


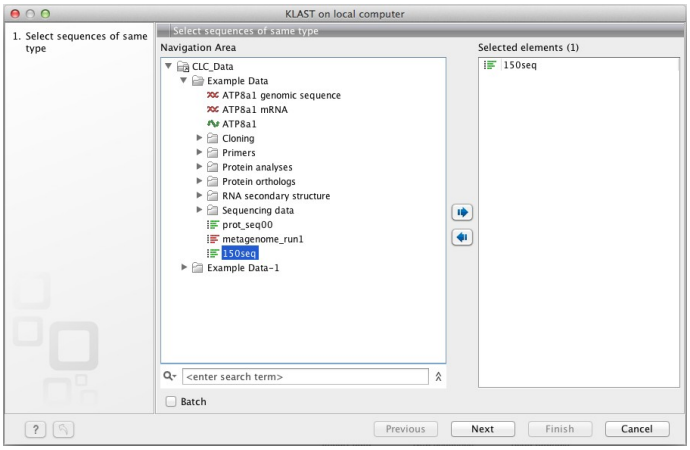
Figure 2 – KLAST algorithm: three step search. Given two sequence databank indexes (query and target), KLAST locates all possible hits (*Subset Seeds* step). From these anchors, the *KLAST filtering* step creates gapped-sequence alignments. Every alignment having a score above a threshold (High-scoring Segment Pairs or HSP) will enter the final step, i.e. the creation of a Smith-Waterman optimal sequence alignment. This last step is called *BLAST like*, since KLAST and BLAST does the same computation at this stage.

3. Setting up a KLAST job

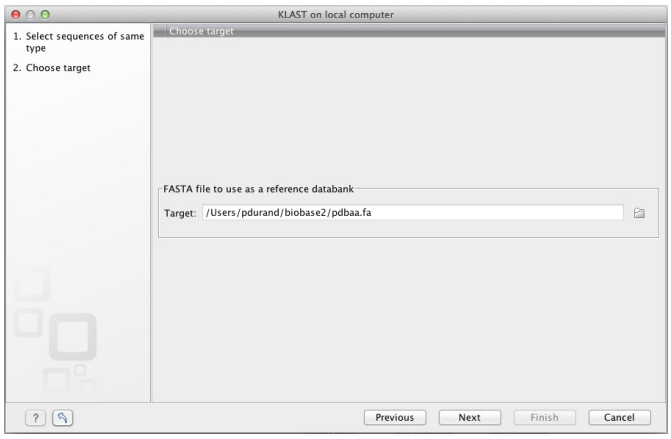
1. From the CLC Workbench's Toolbox, click twice on the « KLAST on local computer » item to start the KLAST job wizard:



2. Select your query sequence from your data set:

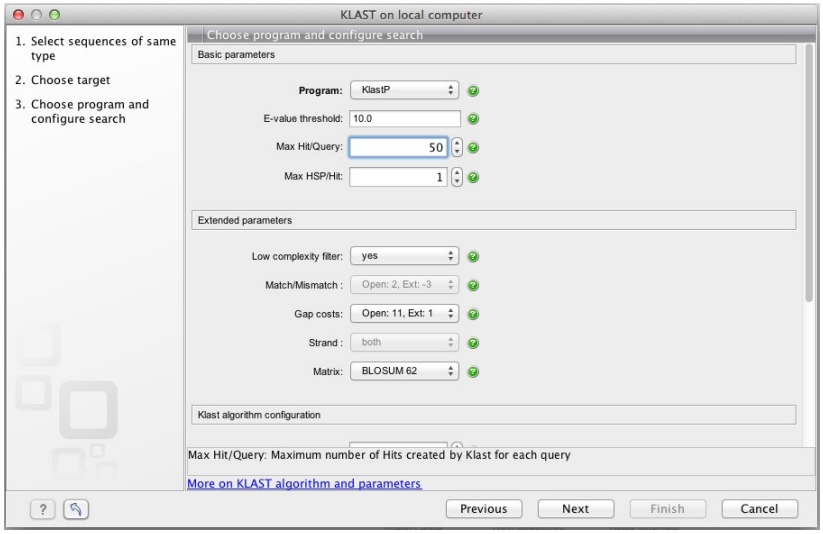


3. Select the target (or reference) databank:



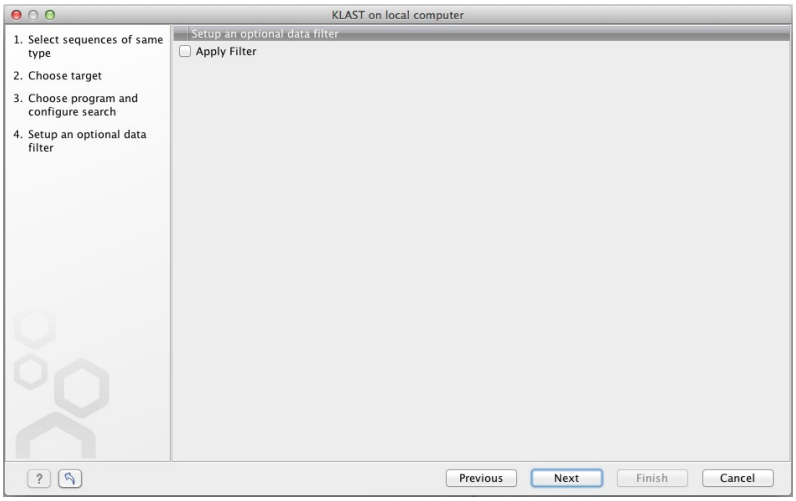
Notice: since KLAST directly uses FASTA formatted sequence file, you have to provide your target databank in such a format. KLAST is not compatible with BLAST binary databanks.

4. Setup the KLAST search parameters.

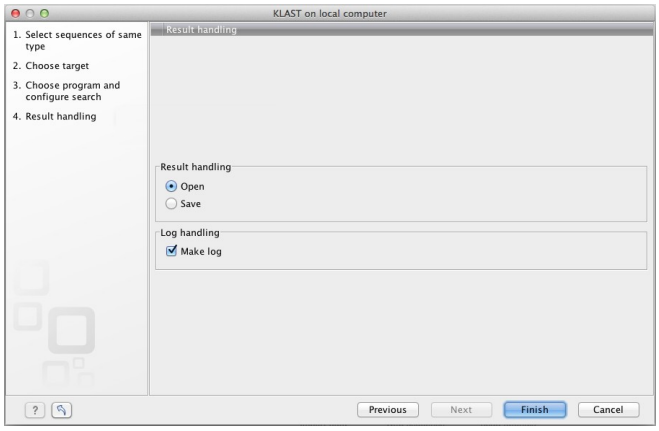


Notice: you have to choose at least the comparison method (klastp, klastn, etc.). KLAST's parameters are detailed in section 6.

5. Choose whether or not to apply a filter. Filters being details on section 7., we let this panel empty in this example:

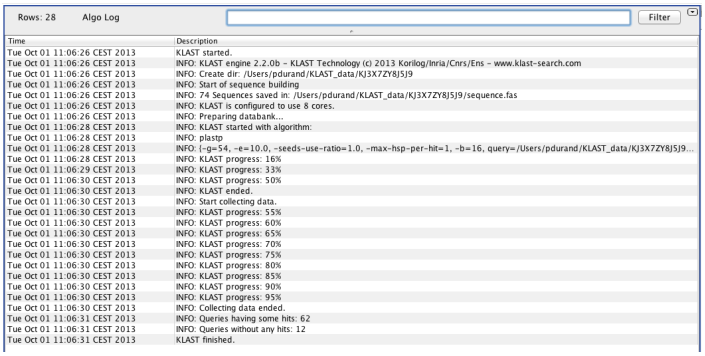


6. Choose the way you would like to see the results; here, we want to get the graphical display (*Result handling = Open*) and ask to see interactive Klast job processing (*Log handling/Make log* is checked) :

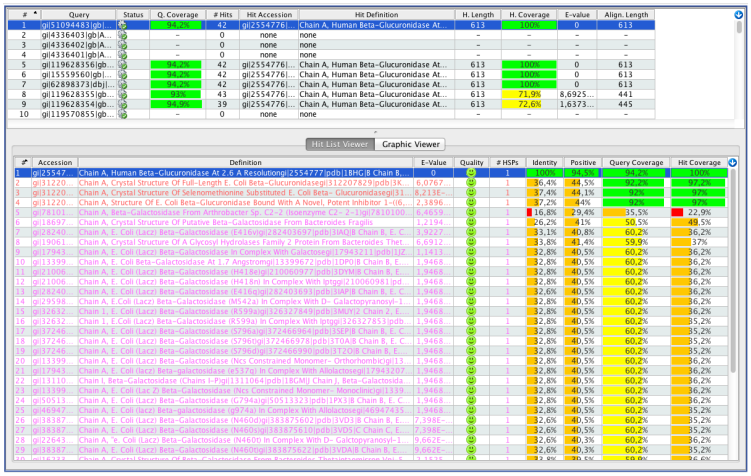


Click on [Finish] to start the job.

7. Monitor data processing during the KLAST job execution:

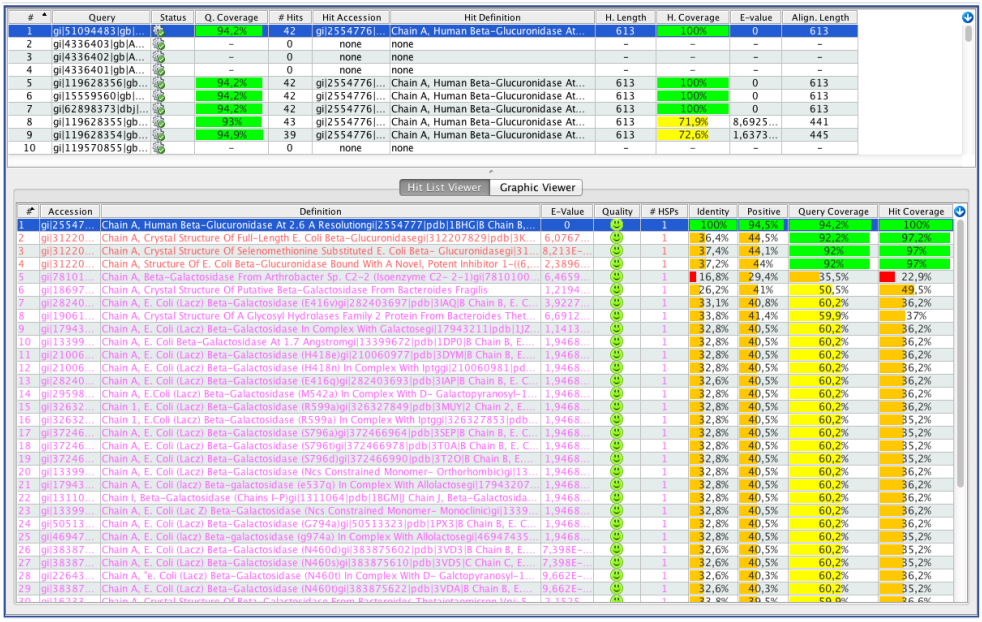

Rows: 28
Algo Log
Filter
Time
Description
Tue Oct 01 11:06:26 CEST 2013 KLAST started.
Tue Oct 01 11:06:26 CEST 2013 INFO: KLAST engine 2.2.0b - KLAST Technology (c) 2013 Korilog/Inria/Cnrs/Ens - www.klast-search.com
Tue Oct 01 11:06:26 CEST 2013 INFO: Create dir: /Users/pdurand/KLAST_data/KJ3X7ZY8J5J9
Tue Oct 01 11:06:26 CEST 2013 INFO: Start of sequence building
Tue Oct 01 11:06:26 CEST 2013 INFO: 74 Sequences saved in: /Users/pdurand/KLAST_data/KJ3X7ZY8J5J9/sequence.fas
Tue Oct 01 11:06:26 CEST 2013 INFO: KLAST is configured to use 8 cores.
Tue Oct 01 11:06:26 CEST 2013 INFO: Preparing databank...
Tue Oct 01 11:06:28 CEST 2013 INFO: KLAST started with algorithm:
Tue Oct 01 11:06:28 CEST 2013 INFO: plastp
Tue Oct 01 11:06:28 CEST 2013 INFO: [-g=54, -e=10.0, -seeds-use-ratio=1.0, -max-hsp-per-hit=1, -b=16, query=/Users/pdurand/KLAST_data/KJ3X7ZY8J5J9...
Tue Oct 01 11:06:28 CEST 2013 INFO: KLAST progress: 16%
Tue Oct 01 11:06:29 CEST 2013 INFO: KLAST progress: 33%
Tue Oct 01 11:06:30 CEST 2013 INFO: KLAST progress: 50%
Tue Oct 01 11:06:30 CEST 2013 INFO: KLAST ended.
Tue Oct 01 11:06:30 CEST 2013 INFO: Start collecting data.
Tue Oct 01 11:06:30 CEST 2013 INFO: KLAST progress: 55%
Tue Oct 01 11:06:30 CEST 2013 INFO: KLAST progress: 60%
Tue Oct 01 11:06:30 CEST 2013 INFO: KLAST progress: 65%
Tue Oct 01 11:06:30 CEST 2013 INFO: KLAST progress: 70%
Tue Oct 01 11:06:30 CEST 2013 INFO: KLAST progress: 75%
Tue Oct 01 11:06:30 CEST 2013 INFO: KLAST progress: 80%
Tue Oct 01 11:06:30 CEST 2013 INFO: KLAST progress: 85%
Tue Oct 01 11:06:30 CEST 2013 INFO: KLAST progress: 90%
Tue Oct 01 11:06:30 CEST 2013 INFO: KLAST progress: 95%
Tue Oct 01 11:06:30 CEST 2013 INFO: Collecting data ended.
Tue Oct 01 11:06:31 CEST 2013 INFO: Queries having some hits: 62
Tue Oct 01 11:06:31 CEST 2013 INFO: Queries without any hits: 12
Tue Oct 01 11:06:31 CEST 2013 KLAST finished.

8. *KLAST Results Viewer* is displayed when the comparison job is done:



4. Analysing results

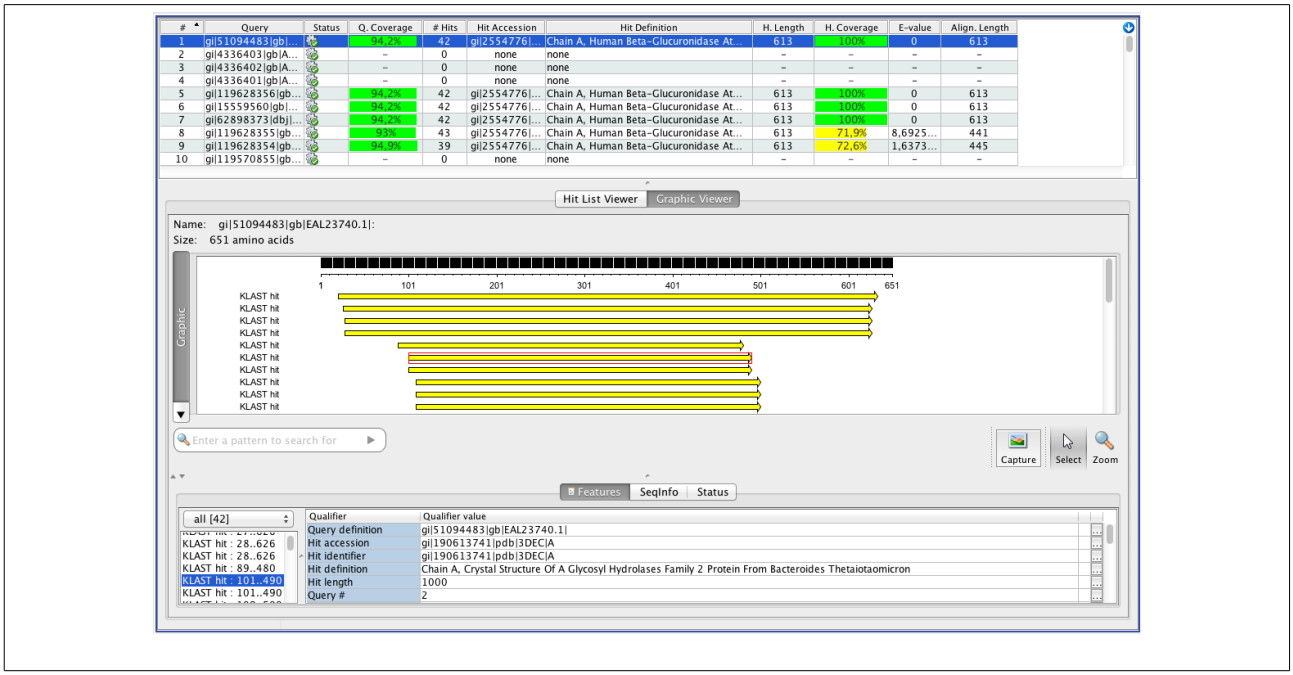
The *KLAST Results Viewer* is divided into two parts:



a. the above table is the best hit table: each row displays a query and its best hit, i.e. the hit with the highest score

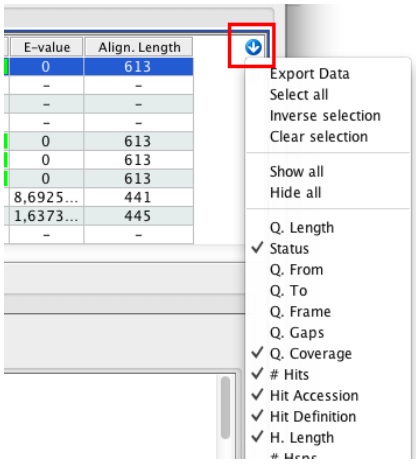
b. the below table is the *Hit List Viewer*: it diplays the whole set of hits reported for a single query.

The *Hit List Viewer* can be replaced by the *Graphic Viewer*, as follows:

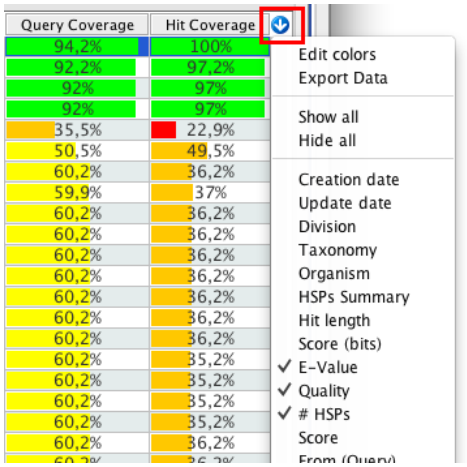


On this view, each hit is represented as a yellow arrow mapped on the query sequence. You can click on an arrow to obtained hit details in the table presented just below the graphic view, and *vice versa*.

Table column content can be changed as needed, using the column selector located at the top right corner of the tables:



Choosing columns for the *Best Hit Table*



Choosing columns for the *Hit List Viewer*

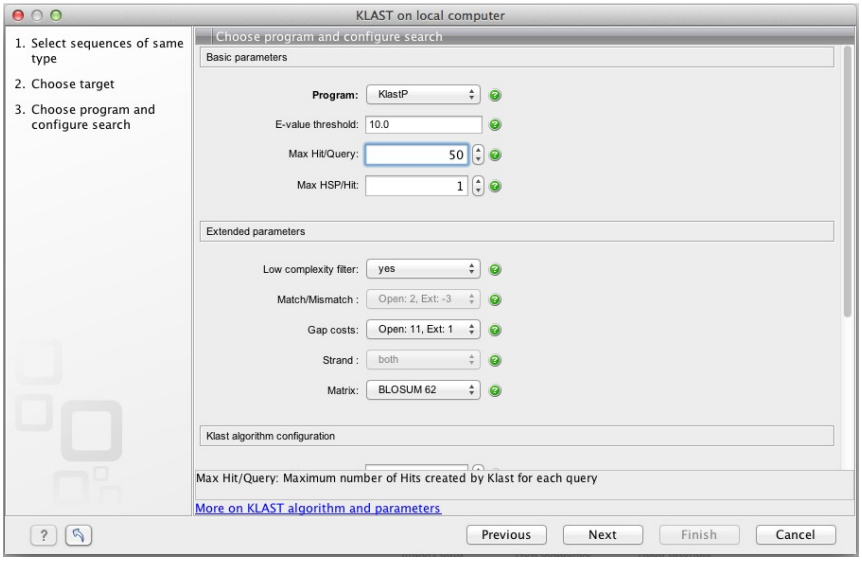
5. Exporting results

Commands to export table content are accessible from the pulldown menu available at the top right corner of the tables (see above pictures).

6. KLAST parameters

Basic search parameters

KLAST parameters are provided by the KLAST Wizard, as follows:



Parameters from sections « Basic » and « Extended » are similar to BLAST ones. However, there are two minor differences. First, comparison method (or *Program*) is one of the following values:

Program	*Description*
KlastP	Search protein database using a protein query
KlastN	Search a nucleotide database using a nucleotide query
KlastX	Search protein database using a translated nucleotide query
tKlastN	Search translated nucleotide database using a protein query
tKlastX	Search translated nucleotide database using a translated nucleotide query

Then, there are two parameters to control how many hits and HSPs are reported in the results:

Program	*Description*
Max Hit/Query	Set the maximum number of hits for each query
Max HSP/Hit	Set the maximum number of HSPs for each hit

Optimizing search job using KLAST specific arguments

KLAST's default configuration has been setup to provide an optimal ratio between speed and quality in order to produce results with quality similar to Blast. Even in such a configuration, you'll have great speedup factors.

However, depending on your needs you can enhance speed factors with little loss of quality in your results.

KLAST specific parameters are provided in section « KLAST algorithm configuration », and three of them are of interest:

Program	*Description*
Percent of seeds to use (or seed-ratio)	Ratio of seeds to be used (see below). [1..100], default is 100. Decrease value to speedup algorithm with little loss of quality. Available for all methods but KlastN.
Threshold score	Ungapped threshold triggering a small gapped extension (see below). [25..127], default is 28 and 55 for protein-based and nucleic-based comparisons, respectively. Increase value to speedup algorithm with little loss of quality.
Maximum DB Size	Maximum allowed size (in bytes) for a databank. If greater, database is segmented (see below).

Fine tuning seed-ratio, threshold score and max-database-size may provide impressive acceleration of the KLAST comparison engine, with little loss of quality in the results. Carefuly read the following sections.

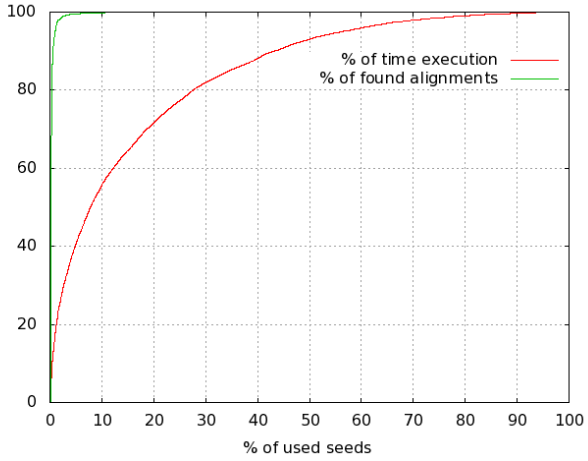
Optimizing KLAST: sample recipes

In order to tune KLAST correctly, we always invite the users to try the software with sample data sets. When you need to compare large set of sequences, always start your work by comparing a small subset of your data. This way, you can check the parameters, the results and the speed of the software.

As an example, if you have to compare 300,000 sequences against NCBI nt, start your work by comparing 300 query sequences against NCBI nt using default KLASTn parameters. Then, fine tune it (see below the use of seed-ratio, max-database-size and threshold score) and check the results. As soon as your parameters are fine, go ahead with 3,000 and/or 10,000 query sequences, and check results and speed. If everything is fine, then run the full comparison.

Optimizing KLAST at runtime: using seed-ratio

When using KLAST for protein-based sequence comparisons, the algorithm can be speedup using the seed-ratio parameter. As stated earlier, KLASTp algorithm relies on a finite table of seeds; there are about 6,200 seeds for BLOSUM50 and BLOSUM62 matrices, whatever the input sequence databanks (for more information, see Reference [3]). During the comparison, KLAST orders seeds by occurrences, starting to process seeds producing the highest number of hits. So, it is possible to ask KLAST to use either the entire set of seeds to achieve a comparison, or a subset. This fine-tuning KLAST feature is achieved using the seed-ratio parameter, ranging from 1% to 100%. The highest seed-ratio you use, the highest sensitivity you get... the lowest seed-ratio you use, the highest speed you get with little loss in quality, as illustrated on this example:



Reducing number of seeds to use during a comparison still provides high results quality while dramatically reducing search time.

The seed-ratio parameter is available for KLASTp, KLASTx, tKLASTx and tKLASTn.

Optimizing KLAST at runtime: using threshold score

A second way to fine tune KLAST, and speedup the search, consists in using the 'threshold score' parameter. During a search, KLAST computes a score for each ungapped sequence alignment matching a query and a hit. As soon as this score is above the threshold, that alignment is retained for further processing. By default, this 'threshold score' is set to a small value (38 for protein comparisons, 55 for nucleotide comparisons) to let KLAST be as sensitive as possible. However, if you suspect that your query sequences may be closely related to the reference databank, you could increase the 'threshold score': KLAST can still produce high-quality results, but with an additional speedup.

As an example, when comparing 900 reads (500 nucleotides on average) against Silva SSU databank (740,000 sequences) on a 8 cores Intel-Xeon based computer, search time was 73 hours using TScore=55, but only 8 minutes using TScore=127; results were the same in terms of quality, i.e. we got the same best hit for each query in both results.

The TScore parameter is available for KLASTn, KLASTp, KLASTx, tKLASTx and tKLASTn. Its value is in the range 25..127 (default is 28 and 55 for protein-based and nucleic-based comparisons, respectively).

Optimizing KLAST at runtime: using max-database-size

Another way to fine tune KLAST, and again speedup the search, consists in using the max-database-size parameter. It sets the amount of bytes to reserve in RAM in order to load databank pages into memory. Indeed, during the comparison of query vs. subject databanks, KLAST automatically paginates databanks if they do not fit entirely into RAM. For that purpose, KLAST relies on the max-database-size parameter; when setting up that parameter, compare the amount of RAM you have in your computer with 'max-database-size x 8 x 2' (each databank index requires '8 x max-database-size' bytes, and you have two databanks). For instance, when using KLAST on a 32 Gb computer, increasing max-database-size from 5M (default value) to 100M may produce an additional speedup of 4x.

We advise you to setup the max-database-size value to enable the full load of the query databank into memory. For instance, if your query file sizes 12 Mb, then set the max-database-size parameter to 15000000 (15 Mb). Also, we do not recommend to set max-database-size to value exceeding 1000000000 (1Gb) ; in such a case, if your query file is very big, let KLAST paginates the query, or run several KLAST jobs, each of them processing one partition of your query file.

The max-database-size parameter is available for KLASTn, KLASTp, KLASTx, tKLASTx and tKLASTn.

7. Filtering KLAST results

Introduction

A data filter, or simply a filter, is made of n>=1 rules. In turn, each rule is defined with a data accessor, an operator and a value. Several rules are combined with boolean operators ‘or’ & ‘and’.

Data accessors

A data accessor enables the access to an information contained in a KLAST result. They are very straightforward, as they correspond to standard sequence comparison tool results:

Accessor name	*Accessor name*
Hit Accession	HSP query from
Hit rank	HSP query to
Number of HSPs	HSP query frame
Hit definition	HSP query gaps
Hit identifier	HSP hit from
Hit Length	HSP hit to
HSP rank	HSP hit frame
Query coverage	HSP hit gaps
Hit coverage	HSP % of positives
HSP bit score	HSP % of gaps
HSP score	HSP % of identities
HSP E-Value	HSP alignment length

Data operators

This table lists the available operators you can use to compare values:

Operator	*Description*	*Accepted values*
==	Equal to	Numeric, string
!=	Not equal to	Numeric, string
<	Less than	Numeric
>	Greater than	Numeric
<=	Less than or equal to	Numeric
>=	Greater than or equal to	Numeric
::	Match	String vs. RegExp
!:	Do not match	String vs. RegExp
[]	Contain in the range	Numeric
][	Not contain in the range	Numeric

All operators but [] and][are equality and relational operators; they are used to compare a left and a right operand.

Operators :: and !: are used for pattern matching to figure out whether or not the right operand is contained within the left operand; left operand is a string, but right operand is a regular expression.

By default, all string-based operators are ‘case sensitive’. However, the *Filter Editor* (see below) enables you to modify this behavior.

Data types

Data handled by Klast are quite simple: integer, real, string (of characters) and date.

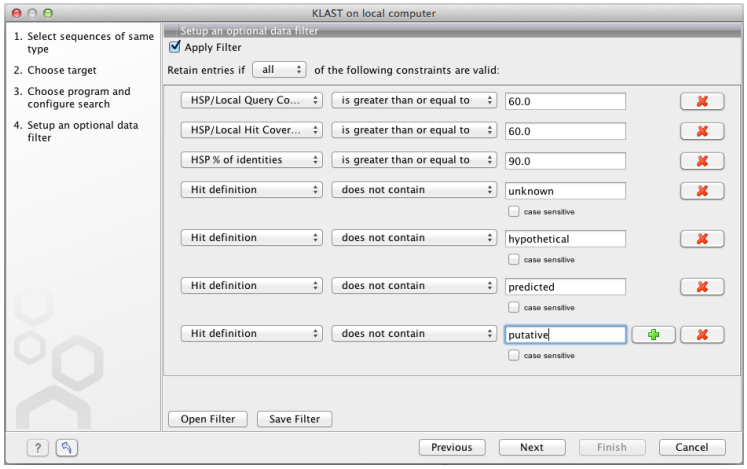
Real numbers can be represented using scientific notation; e.g. 0.001 and 1e-3 are equivalent.

Dates are represented using the following format: YYYYMMDD, where YYYY, MM and DD are the year, month and day, respectively. MM is in the range [01-12] and DD is in the range [01-31]. Example: 20050512 stands for May 12th, 2005.

String can be a litteral string, or it can be a regular expression (see below).

Creating a data filter

A filter is created with the *Filter Editor* during the setup of a KLAST job (step 4). On this panel, simply click on the [Apply Filter] check box to display the edior, then setup your rules as needed:

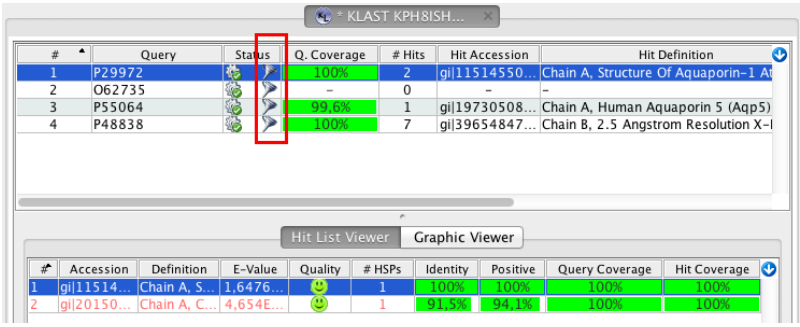

KLAST on local computer
1. Select sequences of same type
2. Choose target
3. Choose program and configure search
4. Setup an optional data filter
Setup an optional data filter
Apply Filter
Retain entries if all of the following constraints are valid:
HSP/Local Query Co... is greater than or equal to 60.0
HSP/Local Hit Cover... is greater than or equal to 60.0
HSP % of identities is greater than or equal to 90.0
Hit definition does not contain unknown
Hit definition does not contain hypothetical
Hit definition does not contain predicted
Hit definition does not contain putative
case sensitive
Open Filter
Save Filter
Previous
Next
Finish
Cancel

Each row corresponds to a single rule. You can add/remove rules using the two command [+]/[x] located on the right side of each rule.

It is worth noting that rules are combined with boolean operators 'or' and 'and' using the 'any' and 'all' items, respectively, displayed in the dropdown list located on top of the rules (see above picture).

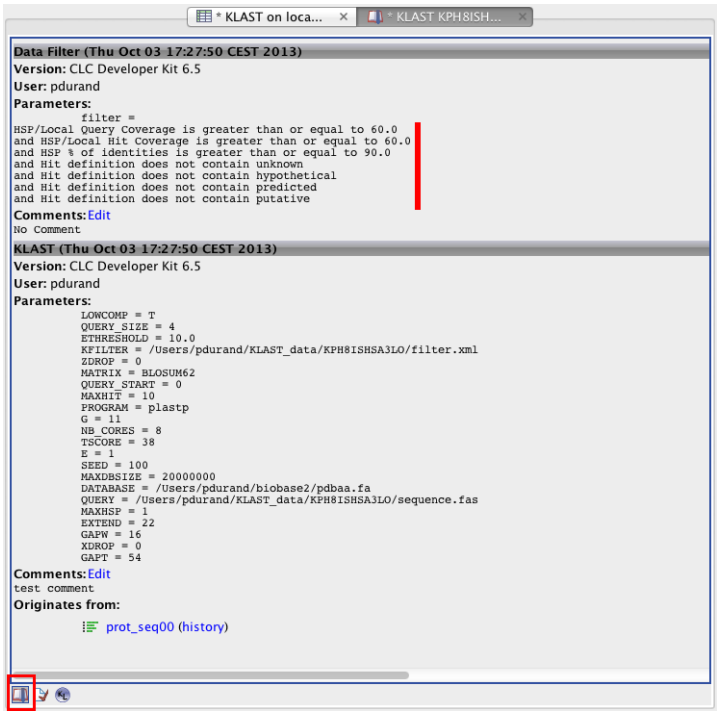
The filter will be applied on the KLAST results automatically during the job execution. Then, the KLAST engine will only report hits validated by the filter rules.

Filtered data can be easily identified on the viewer, as illustrated on this figure:


* KLAST KPH8ISH...
# Query Status Q. Coverage # Hits Hit Accession Hit Definition
1 P29972 100% 2 gi|11514550... Chain A, Structure Of Aquaporin-1 At
2 O62735 - 0 - -
3 P55064 99,6% 1 gi|19730508... Chain A, Human Aquaporin 5 (Aqp5)
4 P48838 100% 7 gi|39654847... Chain B, 2.5 Angstrom Resolution X-
Hit List Viewer
Graphic Viewer
Accession Definition E-Value Quality # HSPs Identity Positive Query Coverage Hit Coverage
1 gi|11514... Chain A, S... 1,6476... 1 100% 100% 100% 100%
2 gi|20150... Chain A, C... 4,654E... 1 91,5% 94,1% 100% 100%

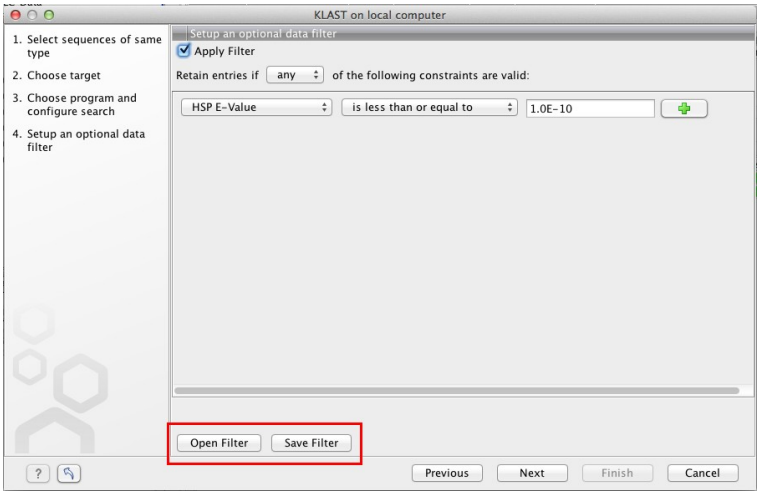
Notice: it is always a good idea to setup and try a filter on a reduced data set in order to validate the rules with regard to the results. Then, go ahead and apply your filter on larger data sets.

To help you identifiying what was the filter used during a particular KLAST job, simply use the job history panel:



Managing filters

The *Filter Editor* provides two commands, [Save Filter] and [Open Filter], to enable you to save and reopen a filter, respectively. This way, you can store and reuse your filters without writing them from scratch.



Using regular expressions

Basic concepts

A regular expression, often called a pattern, is an expression that describes a set of strings. They are usually used to give a concise description of a set, without having to list all elements. Alternatively, a pattern can be used to identify strings containing at least one of the possible strings defined by the pattern. In the field of bioinformatics, there is a well-known example of the usage of regular expressions: the Prosite patterns are nothing else than such expressions.

By the way of operators :: and !:, ngKLAST's filtering system gives you the opportunity to use regular expressions. In the context of filtering KLAST data, these expressions are quite useful to locate a string within another. As an example, if you plan to locate in a KLAST result the database sequences for which the description field contains a particular word, a regular expression will help you, since it allows you to locate a misspelt word. Remember that operators == and != do an exact comparison of two strings: in no way they can be used to locate a particular string within another.

Using a regular expression in a filter

Using a regular expression in a filter's rule is quite easy: as soon as you use one of the operators :: or !:, the right operand (which is the value you specify for the comparison) is considered by ngKLAST as a regular expression.

Regular expression syntax

Very basic expressions

The simplest regular expression is a simple string: A, ATG, kinase, etc.

Alternation

A vertical bar separates alternatives. For example, gray|grey, can match gray or grey.

Grouping

Parentheses are used to define the scope and precedence of the operators. For example, gr(a|e)y, means gr follows by a or e, follows by y. It is equivalent to gray|grey and can also match gray or grey.

Case sensitivity

By default, regular expressions are case-sentitive. As a consequence the pattern atg cannot recognize the string ATG. Using the special construct (?i) allows to search for case-insentitive patterns. The (?i) modifier affects all characters to the right and in the same group, if any. For example in the pattern a(?i)tg, only t is allowed to be case-insensitive, whereas in (?i)(atg) all characters are allowed to be case-insensitive. The pattern (?i)(atg) can match atg, aTg, ATg, ATG, etc.

Quantification

A quantifier after a character or group specifies how often consecutive occurrences of that preceding expression are allowed to occur. The available quantifiers are ?, *, and +.

The question mark indicates there is 0 or 1 of the previous expression. For example, colou?r matches both color and colour.

The asterisk indicates there are 0, 1 or any number of the previous expression. For example, "go*gle" matches ggle, gogle, google, gooogle, etc.

The plus sign indicates that there is at least 1 occurrence of the previous expression. For example, "go+gle" matches gogle, google, gooogle, etc. (but not ggle).

Extended quantification

The expression {x,y}, where x and y are numbers, can be used to define more precise quantification. For example T{x} means T exactly x times. T{x,} means T at least x times. (ATG){x,y} means ATG at least x but not more than y times.

Character classes

Character classes are defined using [and] to match a single character that is contained within the brackets. For example, [abc], matches a, b or c at any given position of a string. Expression [^abc] matches any character except a, b, or c (negation). Expression [a-zA-Z] matches a through z or A through Z, inclusive (range).

Predefined character classes

The following table gives some useful predefined character classes.

Construct	*Matches*
.	Any character
\d	A digit: [0-9]
\D	A non-digit: [^0-9]
\w	a word character: [a-zA-Z_0-9]
\W	a non-word character: [^\w]

Special characters

^ matches the beginning of a string. $ matches the end of a string. For example, ^[hc]at matches hat and cat but only at the beginning of a string, and [hc]at$ matches hat and cat but only at the end of a string.

8. Citing KLAST

Use the following refernce to cite KLAST in your publications: « Nguyen VH, Lavenier D (2009) PLAST: parallel local alignment search tool for database comparison. BMC Bioinformatics. 10:329 ».

9. References

[1] Nguyen VH, Lavenier D: PLAST: parallel local alignment search tool for database comparison. BMC Bioinformatics 2009, 10:329

[2] Lavenier D., Ordered Index Seed Algorithm for Intensive DNA Sequence Comparison, HiComb 2008: IEEE International Workshop on High Performance Computational Biology, Miami, Florida, 2008

[3] Roytberg M, Gambin A, Noe L, Lasota S, Furletova E, Szczurek E, Kucherov G: On subset seeds for protein alignment. IEEE/ACM Trans Comput Biol Bioinformatics 2009, 6(3): 483-494.

[4] https://team.inria.fr/genscale/

Copyright

KLAST is a joint software development by Korilog SARL, CNRS (Centre National de la Recherche Scientifique, www.cnrs.fr), INRIA (Institut National de Recherche en Informatique, www.inria.fr) and ENS (Ecole Normale Supérieure, www.ens.fr).

« KLAST-Korilog Local Alignment Search Tool » is a registered trademark of Korilog SARL.